DOCUMENT RESUME

ED 140 821 IR 004 923

AUTHOR Gentner, Donald R.; Norman, Donald A.
TITLE The Flow Tutor: Schemas for Tutoring. Technical Report No. 7702.
INSTITUTION California Univ., San Diego. Center for Human Information Processing.
SPONS AGENCY Advanced Research Projects Agency (DOD), Washington, D.C.; Office of Naval Research, Arlington, Va. Personnel and Training Research Programs Office.
PUB DATE May 77
CONTRACT N00014-76-C-0628
NOTE 40p.

EDRS PRICE MF-$0.83 HC-$2.06 Plus Postage.
DESCRIPTORS *Artificial Intelligence; *Computer Assisted Instruction; Educational Technology; Individual Instruction; *Programed Instruction; *Programed Tutoring; Programing Languages; Technical Reports; Tutorial Programs; *Tutoring
IDENTIFIERS Automated Tutoring; Flow

ABSTRACT

A human tutor must have a plan of instruction, but be able to deviate from the plan when the student behavior calls for changes. This paper presents observations of human tutors and describes the FLOW tutor system, a computer-based simulation of a human tutor that is capable of giving advice to a student learning a simple computer language. The tutor has a schema-based knowledge structure containing information about the programing language, the student's instruction booklet, and the student's developing knowledge. These schemas form the basis of a distributed intelligence system which uses conceptually guided and data-driven processing to interpret the student's behavior, update the model of the student, and give advice to the student. (Author)

****
* Documents acquired by ERIC include many informal unpublished *
* materials not available from other sources. ERIC makes every effort *
* to obtain the best copy available. Nevertheless, items of marginal *
* reproducibility are often encountered and this affects the quality *
* of the microfiche and hardcopy reproductions ERIC makes available *
* via the ERIC Document Reproduction Service (EDRS). EDRS is not *
* responsible for the quality of the original document. Reproductions *
* supplied by EDRS are the best that can be made from the original. *
****

ED140821

U.S. DEPARTMENT OF HEALTH
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Report No. 7702
May 1977

Donald R. Gentner

Donald A. Norman

THE FLOW TUTOR:

SCHEMAS FOR TUTORING

IR 004923

2

UNIVERSITY OF CALIFORNIA, SAN DIEGO

CENTER FOR HUMAN INFORMATION PROCESSING
LA JOLLA, CALIFORNIA 92093

This research was supported by the Advanced Research Projects Agency and the Office of Naval Research, Personnel and Training Research Programs and was monitored by ONR under Contract N00014-76-C-0628, NR 154-387, under terms of ARPA Order No. 2284. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency, of the Office of Naval Research, or the United States Government.

Approved for public release; distribution unlimited. Reproduction in whole or part is permitted for any purpose of the United States Government.

EA

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 7702	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) The FLOW Tutor: Schemas for Tutoring		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Donald R. Gentner and Donald A. Norman		8. CONTRACT OR GRANT NUMBER(s) N00014-76-C-0628
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Human Information Processing University of California, San Diego La Jolla, CA 92093		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Personnel and Training Research Programs Office of Naval Research (Code 458) Arlington, VA 22217		12. REPORT DATE May 1977 13. NUMBER OF PAGES 27
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Automated tutor, computer-based instruction, frame, individualized instruction, instructional theory, model of the student, representation of information, schema, semantic network, teaching.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A human tutor brings a wide range of knowledge to the task of instructing a student. The tutor must develop a model of the student and of the topic matter. The tutor must have a plan of instruction, but be able to deviate from the plan when the student behavior calls for changes. In this paper we discuss our observations of human tutors and describe the FLOW tutor system, a computer-based simulation of a human tutor that is capable of		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102 LF 014 6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

giving advice to a student learning a simple computer language. The tutor has a schema-based knowledge structure containing information about the programming language, the student's instruction booklet, and the student's developing knowledge. These schemas form the basis of a distributed intelligence system which uses conceptually guided and data-driven processing to interpret the student's behavior, update the model of the student, and give advice to the student.

The FLOW Tutor: Schemas for Tutoring

Donald R. Gentner

and

Donald A. Norman

Center for Human Information Processing
University of California, San Diego
La Jolla, California 92093

Report No. 7702
May 1977

The views and conclusions contained in this document are those of the authors and do not necessarily reflect the policy of any agency of the United States Government. The research was supported by the Advanced Research Projects Agency and the Office of Naval Research and was monitored by ONR under Contract N00014-76-C-0628. The report is approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the United States Government.

The FLOW Tutor: Schemas for Tutoring

Donald R. Gentner and Donald A. Norman

University of California, San Diego

Consider how a human tutor might instruct a student learning a programming language. The student attempts a problem while the tutor watches. The student has difficulty, makes some errors and corrects some errors. Sometimes the tutor gives advice, other times the tutor simply waits for the student to correct the errors without assistance. All through this, the tutor must bring to bear a considerable amount of knowledge. The tutor must have a model of both the student and of the topic matter, simulating the progress of the student through the material to be acquired. The tutor must consider the developing conceptual structures of the student, the student's progress on the current task, and by using some appropriate teaching strategy, decide when it is best to intervene and when it is best to let the student work out the problem alone, without assistance.

Insert Figure 1 about here

Figure 1 shows a sequence of keypresses from a student attempting to solve a computer programming problem. This example

Time	Keypress
1386	Ø
1387	3
1387	3
1390	I
1393	:
1396	RUBOUT
1397	RUBOUT
1400	I
1400	*
1404	Ø
1404	5
1405	Ø
1418	L
1434	R

Figure 1

Sequence of student keypresses on the computer terminal. The number of seconds since the start of the session is indicated in the left column.

shows some of the problems faced by an automated tutor. Based on this information, an automated tutor must decide if the student is making good progress, or if not, what advice to give. Human tutors can deal with this situation. Obviously, the human tutor uses other information to interpret the student's keypresses: information about the programming language and the particular way it is implemented on this computer system, information about the course of instruction and the problem the student is attempting to solve, and information about the knowledge structures of the student. Tutors must also have a knowledge of learning principles to infer just what course of action would be most beneficial for the student: too much advice can be as harmful as not enough. Moreover, the tutor must be flexible. The tutor must have a plan of instruction, but the student may not be ready for that plan. The tutor must be prepared to deviate from the plan whenever the student behavior calls for new tactics. The plan of the tutor constitutes a top-down, conceptually driven guidance of the tutorial session; the behavior of the student constitutes a bottom-up, data driven guidance of the session. A successful tutor must be guided from both directions.

Our goal is to understand the basic cognitive processes involved in learning and teaching, and to develop computer-based theories and models of these processes. Much of the work has been concerned with the learning of a simple computer language, known as FLOW (see Norman, Gentner & Stevens, 1976; Norman,

1975). FLOW is a simple computer language (no subroutines, only one variable). [1] University undergraduates with no previous knowledge of computer programming can usually master the basic elements of FLOW in two-to-ten hours. In the next section of the paper, we discuss our observations of human tutors. Then, we describe the development of a schema-based automated FLOW tutor.

Tutoring

Insert Figure 2 about here

Figure 2 shows the experimental arrangement. A student sits in an acoustically isolated room with an instruction booklet for the FLOW language. The booklet describes computer programming and introduces FLOW with a series of examples and programming problems. The student can try out the examples and attempt to solve the problems on the computer terminal which is connected to a minicomputer. In principle the student could learn FLOW simply by reading the instruction booklet and trying out programs on the terminal. In practice, however, students usually have considerable difficulty and need advice from a human tutor. In our most common tutorial arrangement, the human tutor sits in an adjacent, isolated room where a copy of the student's terminal screen is displayed on a TV monitor. Any advice to the student is relayed via a pair of linked terminals, as shown in Figure 2.

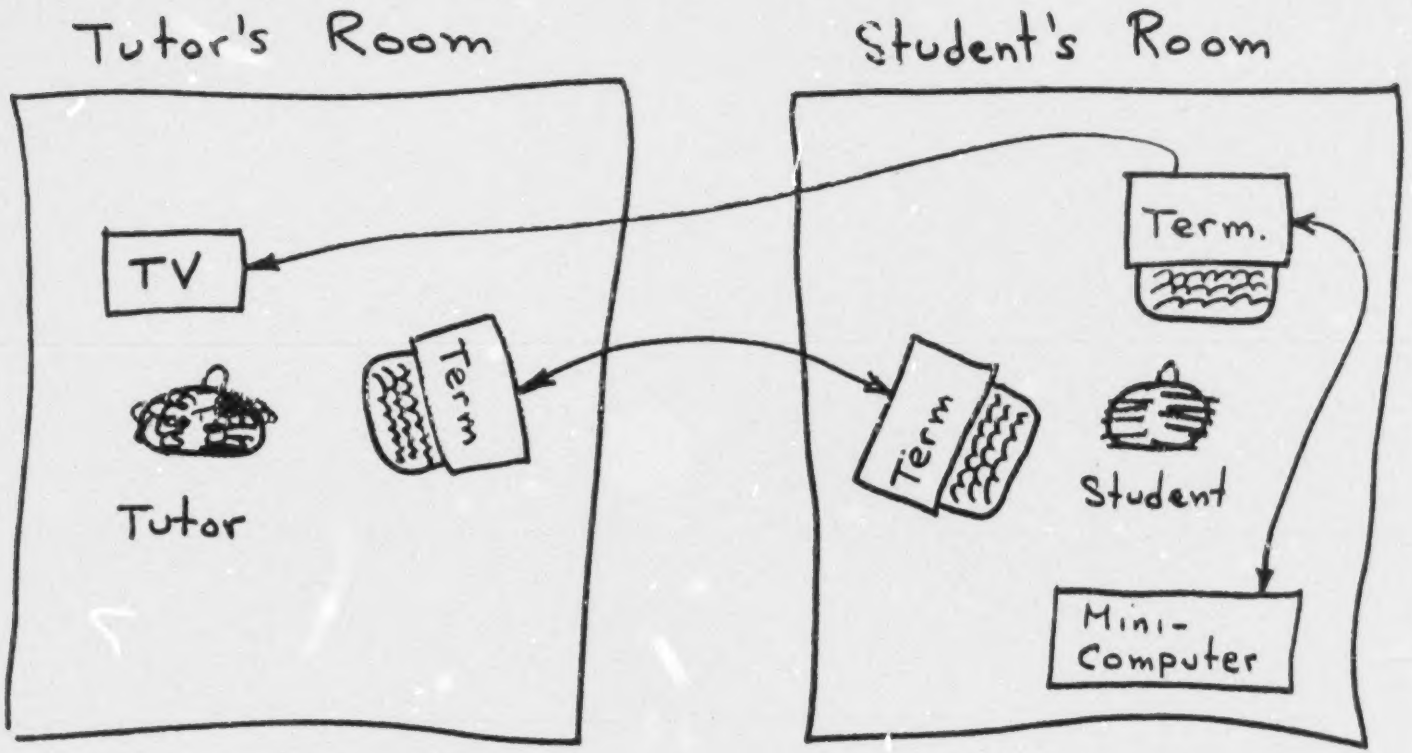

Figure 2

Experimental setup for a student learning FLOW with a tutor in an adjacent room.

Observations of Human Tutors

In our studies of tutorial instruction we have used a variety of techniques. Some have been described previously (Norman, Gentner, & Stevens, 1976; Gentner, Wallen, & Miller, 1974). We examined a number of different dimensions of tutorial interaction and different methods of getting at the conceptual structures of the student:

Tutor continually asking student to think aloud, or simply observing, commenting only when necessary;

A dual-student interaction, with each student helping the other, or simply one student at a time;

Asking the human tutor to simulate an automated tutor, using no more information than would be available to such a system, or letting the human tutor use every source of information possible;

Watching students who have no tutoring at all, but are learning FLOW either in pairs or alone (in either case, with only the instruction manual and an active terminal to guide them);

Asking tutors to have minimum interaction or maximum interaction;

> Retrospective tutoring, in which an actual session is replayed over the computer terminal to the tutor who is asked to treat it as if the student were actually in the adjacent room. The tutor thinks aloud, stating what he thinks the student is doing and what advice he would give, when, and why.

The tutors were all experts at FLOW, and in many cases, experienced teachers. The tutors had before them a copy of the instruction manual being used by the students and they were familiar with its contents (in fact, most of the tutors had been involved in writing the manual).

Some Comments on Experimental Procedure

When the tutor was in a different room from the student, all advice to the student appeared on a second terminal located beside the student. We found it important to use two terminals, in order to make a clear distinction between the tutorial aspects of the session and the workings of FLOW. Earlier, when we had attempted to use the same terminal for both FLOW and tutorial messages, we found that the messages either caused confusion or were overlooked: the students could not distinguish the various sources of information appearing on one terminal. Many students have strange and mystical ideas about how computers work, and the use of a single terminal for both functions added to their confusion. A separate terminal, used only as a tutor, came to be

viewed as the teacher, and students were sometimes recorded talking to it ("well, come on, tell me what to do"). We believe the advantages of using a separate terminal for the tutorial system applies to most topics, including topics not based around programming or use of computers.

We tried various techniques to determine what the student was thinking at each point in the session. One method was simply to have the tutor in the same room as the student, continually asking the student to "think aloud." or "what are you thinking now," or "why did you do that?" In these sessions the students often seemed to be under pressure and responded defensively despite our attempts to be supportive. Our best technique was to ask two students to work together and to put the tutor in a separate room. One student sat at the FLOW terminal and did whatever typing was required. The other student read aloud from the instruction manual. The procedure was quite effective in eliciting the thoughts of the students in natural ways, for they would discuss with each other what they thought each part of the manual meant. When the students encountered problems, they would typically discuss their ideas about the source of the problem, and the possible alternative solutions.

Although we tape recorded the comments of the students, we usually did not let the tutor hear them. Later, we could replay the session, allowing us to evaluate the tutor's hypotheses. Often the students' comments showed that the tutor had misjudged

the problems faced by the students. We recommend this two-student instructional dialog technique as a way of getting naturalistic protocols of student thougnts. Of course, it is awkward to attempt to simulate a two-headed student, so for some purposes, the technique cannot be used. (We have considered using a trained experimenter to play the role of the second student, thus eliciting natural comments without the pressure of a tutor's prying. There are many difficulties far with this scheme, however, and so we have not used it.)

Six Principles of Human Tutoring

Our observations of human tutors are hard to quantify, but they did provide us with useful principles with which to guide the development of the automated tutor. Human tutors are guided in a variety of ways. The following six principles emerged from our observations.

1: Conceptually driven guidance. Tutors have a plan of instruction which sets the overall structure for the session and helps guide their expectations of student performance. This is top-down or conceptually driven guidance. Tutors normally differ in how well formulated these instructional plans are, but in our experiments the instruction manual was provided, so this aspect of tutoring was determined beforehand.

2: Data-driven guidance. Tutors are responsive to student behavior. Thus, they can be data-driven, deviating from the

lesson plan whenever it seems sensible to do so. Data-driven tutoring seemed always to be triggered by errors (which includes a failure to make any response) or by unexpected correct responses. The response of the tutor to a student error seems best characterized as an attempt to find some explanation for the student's behavior, then postulating some instructional sequence that will overcome the problem. Thus, the events witnessed by the tutor trigger a search for an adequate conceptualization that can serve as an explanation for the observations. Once formed, the conceptualization then acts as top-down, conceptual guidance for the lesson plan until the tutor is satisfied the problem is overcome. Then the original plan (from the instruction manual) is resumed.

3: Active discovery. Tutors seemed unwilling to interfere too often. Their method of instruction seemed to contain an implicit assumption that it was best for the students to discover the concepts through active exploration, which includes making numerous mistakes. Tutors would therefore allow students to make errors or to pause for rather long periods of time, offering help only if the number of mistakes or length of pause exceeded some threshold value of tolerance. There were exceptions to this policy. If an important piece of information seemed to be entirely missing then it would be offered directly (to minimize time and frustration). Similarly, if the problem seemed unimportant, it didn't seem worth the student's effort to let them worry about

it, and the student was simply told what to do (for example, that to type a quote mark, you must type a "shift-2"). Tutors also worried if the student seemed to have several simultaneous misunderstandings. The ideal situation for discovery learning seems to be when the student has a single incorrect concept and the resources to locate and modify that concept in a reasonable amount of time.

4: Say too little, not too much. There seemed to be an implicit feeling among several of the tutors that it was best to err by saying too little rather than too much. This aspect of tutoring varied considerably among the tutors, however. This is consistent with the principle stated above -- discovery learning. Thus, the information given the student often was in the form of guides or small fragments of information that would allow the students to discover for themselves the total story. Sometimes, the advice was to re-read a section of the manual. This form of advice giving can be characterized as tutoring by offering clues rather than tutoring by giving answers.

5: Wait and see. Tutors often hung back in their assessment of student difficulty. This was revealed most dramatically when we asked tutors to watch the replay of previous learning sessions. We wanted to study how tutors updated their model of the student, so we probed the tutors after each student response (or after each 11 seconds of pause). Tutors would often refuse to speculate, claiming that they wanted to wait and see what would

happen next. We have decided that this refusal to speculate after each item of student behavior is quite beneficial, for it limits the amount of needless search and analysis the tutor needs to do. By waiting for a reasonable sequence of responses, the number of possible interpretations is considerably constrained, and the task of forming a conceptual model of student performance is much simplified. This strategy takes advantage of the fact that it is not necessary to correct a student problem the instant it is detected.

6: A little irrelevancy is ok. Tutors discovered that they did not always need to be accurate in their assessment of student behavior. Oftentimes a puzzling sequence of behavior would turn out to be irrelevant, for the student would get on the appropriate track without aid. The tutor could afford to ignore things. Even if retrospective analysis showed the behavior to be a hint of forthcoming difficulties, little was lost by not noting it immediately. If there was a real difficulty, students would reveal it again. Even when tutors misinterpreted student behavior and thereby offered irrelevant advice, little harm was done: the students seemed quite content to ignore the advice. (Unfortunately, they frequently seemed content to ignore relevant advice, too.)

The above comments on tutorial strategies should not be over interpreted. Our tutors did not work independently of one another. They consisted of the several of us who worked on the

FLOW project. We often discussed our strategies. Sometimes several tutors would work together, so the instruction was a joint venture of two or more people. Even when a single tutor seemed to provide useful information, we have no way of knowing if this was an optimal instructional sequence. All we can say is that we have observed ourselves tutoring students, and we have made six generalizations from these observations. We will use these principles in the design of the automated tutorial system. But the studies are of real tutors, not ideal tutors.

The Automated FLOW Tutor

The automated FLOW tutor is designed to simulate a human tutor. From now on, unless we explicitly say "human tutor," the word "Tutor" will refer to the computer program designed to serve this function. The Tutor receives a message from the student's minicomputer whenever the student presses a key on the terminal or pauses for more than about eleven seconds. The automated Tutor must interpret these keypresses and pauses in terms of a student progressing through the FLOW instruction booklet. If the Tutor determines that the student is having difficulty, the Tutor can send advice which appears on a second terminal in the student's room.

Ambiguities

Student responses are often ambiguous. This is especially true where the only information available to the tutor is the

time sequence of keypresses made by the student. This, of course, is the only information available to the automated Tutor. Ambiguities can take many forms. For example, in isolation, the individual keypresses are often ambiguous. In the FLOW computer system, when the student types an illegal character, it is displayed briefly accompanied by an audible "beep," and then it is erased. Determining why that character was typed can be difficult. But even legal characters are often ambiguous. In FLOW, for example, a D keypress may be part of a Display Quoted String statement. In other circumstances, a D keypress could also be part of a Display Variable statement or part of a character string. Similarly, a two-minute pause might reflect the fact that the student was completely frustrated and needed immediate help. But a two-minute pause could also occur while the student was reading the instruction booklet and making perfectly normal progress. The task for the Tutor, then, is to construct a broad and detailed model of the student's conceptual understanding and activities and use that model to interpret the current behavior of the student. This interpretation of the student's behavior must then be used to continually update the model of the student.

Conceptually Guided Processing

The normal method used to disambiguate student behavior is conceptually guided prediction. The automated Tutor follows the student's progress through the instruction booklet, allowing for pauses while the student is reading. When an exercise or

programming problem is encountered, the Tutor does the exercise or solves the problem and predicts that the student's actions will follow a similar course.

The Tutor's database contains a description of the function of each problem presented to the student. The Tutor "solves" the problem by expanding the function description into simpler functions, then into FLOW statements, and finally into the individual keypresses. [2] For example, suppose that a problem at some point requires that the computer display the word "BILLY" on the screen. The Tutor will eventually predict a Display Quoted String statement and that the student will press the D key. If the student actually presses the D key at that point, that D will be interpreted as part of a statement to display "BILLY," even though it is perfectly possible at this point that the D is part of some other statement. After observing the D key, the Tutor will then go on to predict that the student will press a quote key, then a B key, and so forth. Only if these later predictions are not confirmed, will the Tutor consider other possibilities for the initial D key. Although it is clear that conceptually guided prediction can lead the Tutor into severe trouble when the predictions are wrong, it normally leads to an easy and efficient interpretation of otherwise ambiguous information.

Data-Driven Processing

Conceptually guided processing works well as long as the student's actions match the predictions of the Tutor. Unfortunately, students often do unexpected things. Remember, in our observations of human tutors, we found that they were sometimes unable or unwilling to make detailed predictions of student behavior, especially when there was more than one reasonable alternative for the student. To simulate this behavior, the FLOW Tutor normally stops the conceptually guided expansion of instances when an ambiguity is reached, and waits to see what the student actually does before going further. If we call the conceptual structures that the Tutor uses to understand the student schemas, then the job of the Tutor is that of creating appropriate schemas to explain student behavior.

The automated FLOW Tutor is intended to simulate an experienced human tutor who is familiar with FLOW and the problems students commonly have when learning it. The Tutor's starting database thus has schemas which represent typical student errors. With data-driven processing, unexpected student inputs may incorporate themselves into instances of these error schemas. When one of these error schemas is fully satisfied, the Tutor has effectively recognized a student error, and can act accordingly. The general strategy of the Tutor is to let the student recover from errors on their own, and only give advice when the student is likely to become frustrated or confused. Thus, when an error

schema discovers a student error, it normally predicts that the student will pause, and only if that pause is actually observed will it give advice to the student. If the student does something before the end of the predicted pause, the error schema will be unsatisfied and disappear. The latest student action will initiate new schemas which will take over the processing. The pause length predicted depends on the type of error and the Tutor's model of the student. In general, if the Tutor believes that the error is related to a concept which is new to the student, advice will be given after a relatively short pause. On the other hand, if the student has previously used this concept correctly, the Tutor will allow a longer pause before giving advice.

Schemas

The FLOW tutorial system is based upon a schema representation for knowledge of both the declarative aspects of the data base and the procedural aspects of tutoring strategy. Schema- or frame-based systems for the representation of information have been proposed by numerous workers in Psychology and Artificial Intelligence, but there have been few attempts to use them as the basis for working systems (for an example of a related schema-based system see Bobrow, Kaplan, Kay, Norman, Thompson, & Winograd, 1976).

Insert Figure 3 about here

The top left portion of Figure 3 shows a fragment of a FLOW program, consisting of two Display Quoted String statements. Each statement has a statement number on the left and the character string to be displayed is included between the quotes. The student only has to type the underlined characters; the other characters are automatically provided by the computer. When these statements are executed, the words BILLY and JEAN would be displayed on the terminal screen.

The general schema for a Display Quoted String statement is shown in the bottom left portion of Figure 3. The notation follows that of the MENOD semantic network system described in Norman, Rumelhart & the LNR Research Group (1975). The schema has a name, on the top line, followed by a series of slots, one on each line. The first two slots are "argument" slots, which are used to distinguish individual instances of the schema, and are thus EMPTY in the generic schema. (Our schemas typically have one to three arguments.) The "specialist" slot gives the name of a procedure, in this case DISPLAYQSER, associated with the schema. The "pnost" slot will be discussed later. The last two slots in the schema give particular instances of this schema.

Two instances of the DISPLAY-QUOTED-STRING schema, corresponding to the two statements in the upper left portion of the figure, are shown on the right in Figure 3. The first slot

```
020 DISPLAY "BILLY"

030 DISPLAY "JEAN"
```

```
DISPLAY-QUOTED-STRING
   statement-number EMPTY
   value EMPTY
   specialist DISPLAYQSER
   phost DISPLAY
   instance *DISPLAY-QUOTED-STRING-1786
   instance *DISPLAY-QUOTED-STRING-1932
```

```
*DISPLAY-QUOTED-STRING-1786
   schema DISPLAY-QUOTED-STRING
   statement-number 020
   value BILLY
   status OBSERVED
   host *DISPLAY-1853
   element *D-1796
   element *QUOTED-STRING-1805

*DISPLAY-QUOTED-STRING-1932
   schema DISPLAY-QUOTED-STRING
   statement-number 030
   value JEAN
   status OBSERVED
   host *DISPLAY-1911
   element *D-1937
   element *QUOTED-STRING-1941
```

Figure 3

The schema formalism.

on these instances points back to the original schema. This is followed by the two argument slots, which are now filled in. Next comes a "status" slot, which indicates that this instance has been OBSERVED. Schemas and instances are composed of other schemas and instances, and that structure is reflected in the final three slots. The "host" slot points to a higher level instance which this instance is part of. In these cases, the instance is part of a DISPLAY instance. Conversely, the "element" slots point to the lower level instances which make up this instance. Here we see that each instance is composed of an instance of a D schema and an instance of a QUOTED-STRING schema. An instance may have several elements, but is restricted to a single host.

Insert Figure 4 about here

Figure 4 shows the host and element instances of the second DISPLAY-QUOTED-STRING instance shown in Figure 3. The DISPLAY instance describes the function of the DISPLAY-QUOTED-STRING instance, namely to display JEAN. It in turn is part of a more complex DISPLAY-SEQUENCE instance.

The D instance shown in the figure has a single argument slot, giving the time when the student pressed this key. Keypresses and TIME messages are the lowest level schemas used by the FLOW Tutor, and therefore the D instance does not have any elements. The QUOTED-STRING instance shown on the right also has

```
*DISPLAY-1911
   schema DISPLAY
   after *DISPLAY-1853
   value JEAN
   status OBSERVED
   host *DISPLAY-SEQUENCE-1842
   element *DISPLAY-QUOTED-STRING-1932

*D-1937
   schema D
   time-observed 00857
   status OBSERVED
   host *DISPLAY-QUOTED-STRING-1932
```

```
*QUOTED-STRING-1941
   schema QUOTED-STRING
   value JEAN
   status OBSERVED
   host DISPLAY-QUOTED-STRING-1932
   element *QUOTE-1946
   element *CHARACTER-STRING-1951
   element *QUOTE-1955
```

Figure 4

The host and elements of an instance.

a single argument slot, giving the value of the string inside the quotes. This instance is further decomposed into elements: two instances of QUOTE and an instance of CHARACTER-STRING.

All the instances which the FLOW Tutor creates and works with are part of a multi-level structure. The highest level instances correspond to such things as the instruction booklet, programming problems, and the function of programs. The lowest level instances correspond to the individual FLOW statements and keypresses.

This hierarchical structure of instances plays a major role in the operation of the FLOW Tutor. Extension of the hierarchy to higher and lower levels forms the basis for predicting and interpreting student behavior. The hierarchical structure gives the Tutor multiple descriptions of the same information at different conceptual levels. Student actions can then be dealt with at levels appropriate for both the Tutor and the student.

As a schema-based system, the FLOW Tutor has an inherently distributed intelligence. That is, there is no central process which is "aware" of everything at a high level and controls its subprocesses. Instead, each schema knows only about the things which immediately concern it. When a schema and its instances become active they try to find their parts and fit themselves into higher level schemas. The only central coordination is furnished by an agenda, a simple list of instances waiting to be

active. Distributed intelligence systems such as this are based on the premise that a large number of relatively simple, independent processes, each concerned only with itself and its immediate environment, will have the net effect of a powerful intelligence.

How Schemas Operate

When an instance becomes active in the FLOW Tutor system, the specialist for its schema is invoked. The specialist can then act based on an examination of the instance and any relevant parts of the Tutor's model of the world. There are several types of actions the specialist can perform. The specialist may modify an instance, predict new instances of schemas, change the Tutor's model of the world, look for input from the student, put instances on the agenda, or send messages to the student. In a typical case, an instance on the agenda might have been predicted by some other instance. When the instance becomes active, its specialist would check to see if all of its elements had been observed. If not, the specialist would predict the next element and put it on the agenda. If all the elements of the instance had been observed, the specialist would change the status of the instance to OBSERVED and place its host on the agenda. If the instance had no host, the specialist might search for a host and try to incorporate the instance into a suitable higher level instance.

When a student responds in a manner not predicted by the conceptually guided analysis, the Tutor starts processing in a data-driven manner. When this happens, the Tutor stops the conceptually-guided expansion of instances and waits to see what the student does. New instances are made up by lower level instances or keypresses, which are initially without a host. These instances try to find a suitable host, or create one, and the resulting structure builds up until it joins some of the existing higher level structure. There are two situations in which data-driven processing is particularly interesting: with alternate correct solutions, and with student errors.

Alternate correct solutions. Although the programming problems in the FLOW instruction booklet are rather elementary, most of them have many different acceptable solutions. When the student enters a program different from that predicted by the Tutor, data-driven processing is used to incorporate the keypresses, statements, and simple functions into a representation for the overall function of the program. Information in a schema representation is enmeshed in a hierarchy, and two programs which may be completely different at the keypress or statement level can be equivalent when compared at higher levels which represent simple or complex functions.

Two instances are considered formally equivalent if they are instances of the same schema and have the same argument values; they may have different elements and still be equivalent. Thus

two function instances might be formally equivalent even though they were composed of completely different FLOW statements. A simple example of this can be seen by comparing the instances of DISPLAY-QUOTED-STRING and DISPLAY in Figures 3 and 4. The DISPLAY-QUOTED-STRING instance is distinguished in part by its statement number, but the corresponding argument slot for the DISPLAY instance gives its functional sequence in the program. Thus the DISPLAY-QUOTED-STRING instance for the statement

Ø26 DISPLAY "JEAN"

would be distinguished from *DISPLAY-QUOTED-STRING-1932 because of their different statement numbers, but at the level of a functional description the two DISPLAY instances would have the same arguments (both are after *DISPLAY-1853) and would be formally equivalent.

Handling a Student Error

Insert Figure 5 about here

Figure 5 shows examples of error schemas operating within an actual student protocol. The student has written a program and is now about to modify it to eliminate an error. The protocol in Figure 5 begins as the student is finishing a reading pause. The student must list the program before it can be modified, and the Tutor has therefore predicted that the student will press the L key (for the List command). Instead of pressing the L key, however, the student presses the RUBOUT key. RUBOUT is an

```
00946 TIME
00954 RUBOUT
00965 TIME
00976 TIME
00987 TIME
TUTOR: TO MODIFY YOUR PROGRAM YOU MUST FIRST
       LIST YOUR PROGRAM
00992 RUBOUT
00999 R
01010 TIME
01011 RUBOUT
01022 X
01024 D
01035 TIME
01046 TIME
TUTOR: TO LIST YOUR PROGRAM YOU MUST PRESS
       THE L KEY
01058 L
01068 TIME
01079 TIME
01083 RUBOUT
01088 RUBOUT
01094 1
01096 5
01102 SPACE
01113 TIME
01116 D
```

Figure 5

Protocol of an automated tutorial session. (Lines preceded by a number correspond to student keypresses. The TIME message indicates that the student has not pressed any keys since the last message.)

illegal key in this context, and was not predicted by the Tutor. The Tutor now uses data-driven processing to interpret the unexpected RUBOUT key. An instance of RUBOUT is created, and it tries to find a suitable host for itself. Each schema contains information about possible hosts (in the "phost" slot), and RUBOUT makes a breadth-first search up through the hierarchy looking for an instance which has been predicted. RUBOUT eventually finds that it could be part of a predicted MODIFY-PROGRAM instance, but MODIFY-PROGRAM had predicted LIST as its next element. Therefore RUBOUT instantiates the INCORRECT-ORDER schema on the assumption that the student was trying to modify the program, but forgot to list it before using RUBOUT to change the statement number. As is typical of error schemas, the schema for an INCORRECT-ORDER error includes an expected pause. In general, the pause lengths predicted depend on the the type of error and the student's knowledge of the the relevant concepts, with shorter pauses allowed for concepts which are newer to the student. In this case, since the Tutor's model of the student indicates that this student has already used the List command, a moderately long pause of 30 seconds is predicted. When the 30-second pause is observed, the INCORRECT-ORDER instance is satisfied, and the Tutor sends a message advising the student to list the program. Having just told the student to LIST, the Tutor now quite naturally expects the student to press the L key to list the program. The student, however, tries several other keys (most of them illegal) and finally, after another, shorter

pause, the Tutor gives more explicit advice to the student.

Current Status of the FLOW Tutor

The automated FLOW Tutor system currently operates only in "retrospective mode." Protocols are recorded as students learn FLOW, with a human tutor in another room simulating the automated Tutor. (Figure 5 is an example of such a protocol.) Selected portions of these protocols then serve as input for the automated Tutor. The system does not yet have a sufficient data-base to follow a student through the entire FLOW course. In addition, it is too slow to respond in real time. Thus, the system has not yet been used with real students, although this is our eventual goal. To paraphrase Bobrow et al. (1977), the FLOW Tutor does realistic tutoring, but it does not yet do real tutoring.

Summary

Our goal is to develop a theory of learning and teaching. In particular, we are interested in the process of individualized instruction and in the type of interactions which take place between a student and a teacher. In this paper we describe a preliminary step towards the development of such a theory. Here, we use the observations of numerous tutorial sessions between human tutors and students to characterize the tutorial process. Then, we show how these ideas can be combined with a schema-based representational system to form an automated Tutor.

Three sources of knowledge must be used in succesful tutoring. First, there must be knowledge about the subject matter. Second, there must be knowledge of the student, including some estimation of the current state of knowledge that the student has about the topic being studied. Third, there must be knowledge of the principles of learning, so that appropriate tutorial intervention can take place.

A successful model of a tutor must therefore contain all these types of knowledge. In addition, the process structure of the tutorial system must allow it to be both conceptually guided and data-driven, depending upon the performance of the student. This paper serves both as an outline of some of the properties of tutorial interaction and also as a case study of the development of a working automated tutorial system.

The automated system cannot now instruct real students. At the moment, the knowledge and processing structures required for succesful tutorial instruction in the wide variety of situations which can occur are more complex than can be handled. But we believe the deficiencies are ones of quantity, and not of basic principle. As computers become cheaper and more powerful, it becomes feasible to think of small, independent, automated teaching systems which could have a real understanding of their subject matter and interact intelligently with students having different backgrounds and abilities. Before such systems can be

built on more that a limited, ad-hoc basis, we must learn a great deal more about learning, teaching, and the representation and organization of knowledge. The FLOW Tutor project is directed towards these goals.

References

Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., & Winograd, T. GUS, a frame driven dialog system. Artificial Intelligence, 1977, 8, in press.

Gentner, D. R., Wallen, M. R., & Miller, P. L. A computer-based system for studies in learning (Tech. Rept.). La Jolla, Calif.: University of California, San Diego, Center for Human Information Processing, September 1974.

Norman, D. A. Learning and teaching. In P. M. A. Rabbitt & S. Dornic (Eds.), Attention and performance V. Proceedings of the Fifth Symposium on Attention and Performance, Stockholm, Sweden. London: Academic Press, 1975.

Norman, D. A., Gentner, D. R., & Stevens, A. L. Comments on learning: Schemata and memory representation. In D. Klahr (Ed.), Cognition and instruction. Hillsdale, N. J.: Erlbaum Associates, 1976.

Norman, D. A., Rumelhart, D. E., & the LNR Research Group. Explorations in cognition. San Francisco: Freeman, 1975.

Raskin, J. Flow language for computer programming. Computers and the Humanities, 1974, 8, 231-237.

Footnotes

The research was supported by the Advanced Research Projects Agency and the Office of Naval Research of the Department of Defense and was monitored by ONR under Contract No. N00014-76-C-0628. Mark Wallen made significant contributions to the research -- programming and maintaining the FLOW system and associated tutorial and analysis systems, and acting as tutor on numerous occasions.

1. FLOW was originally developed by Jef Raskin of the Visual Arts Department of the University of California, San Diego, specifically for students with no mathematics or science background; see Raskin (1974).

2. The programs required of the student are all conceptually very simple -- the most complex problem given to the student is: "Write a program which will display the word 'yes' if the input text contains an E and 'no' otherwise." Thus, the usual difficulties of writing a program from a problem statement (automatic programming) do not arise.